



1/16

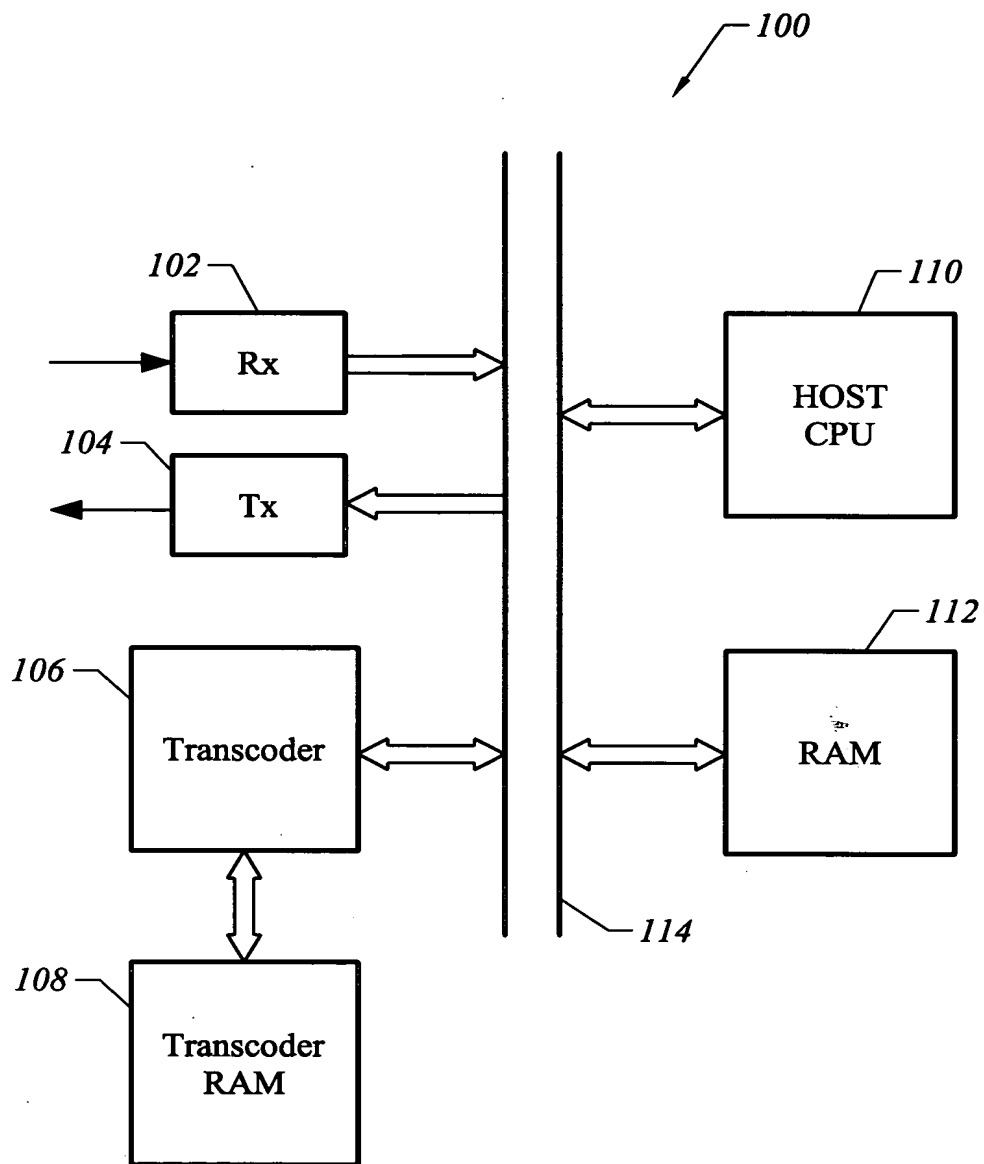


FIG. 1

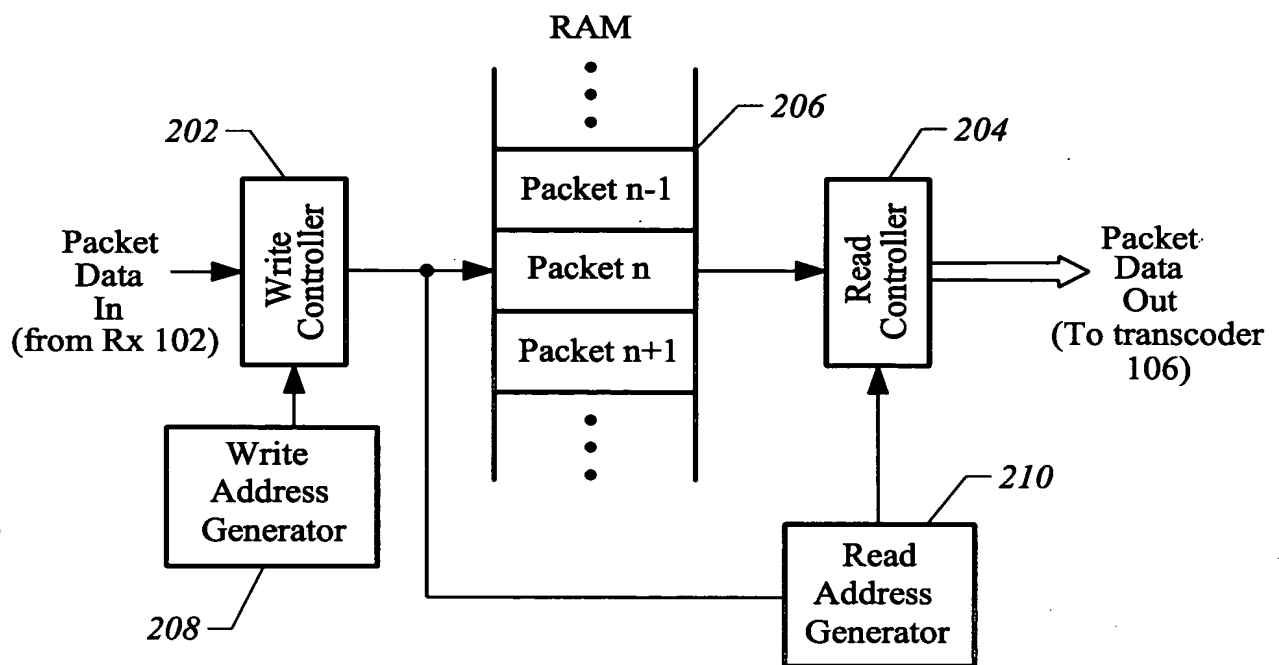


FIG. 2

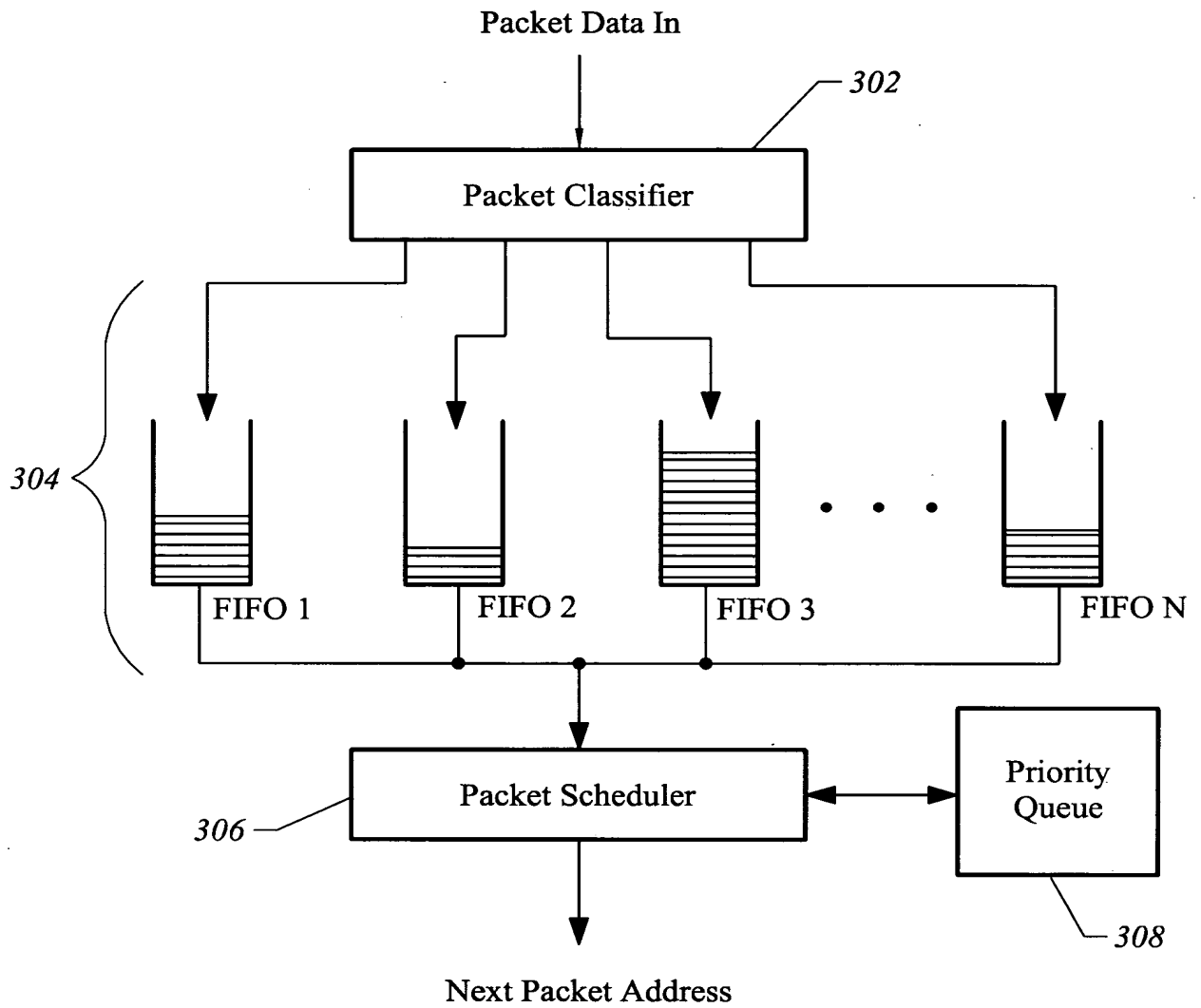


FIG. 3

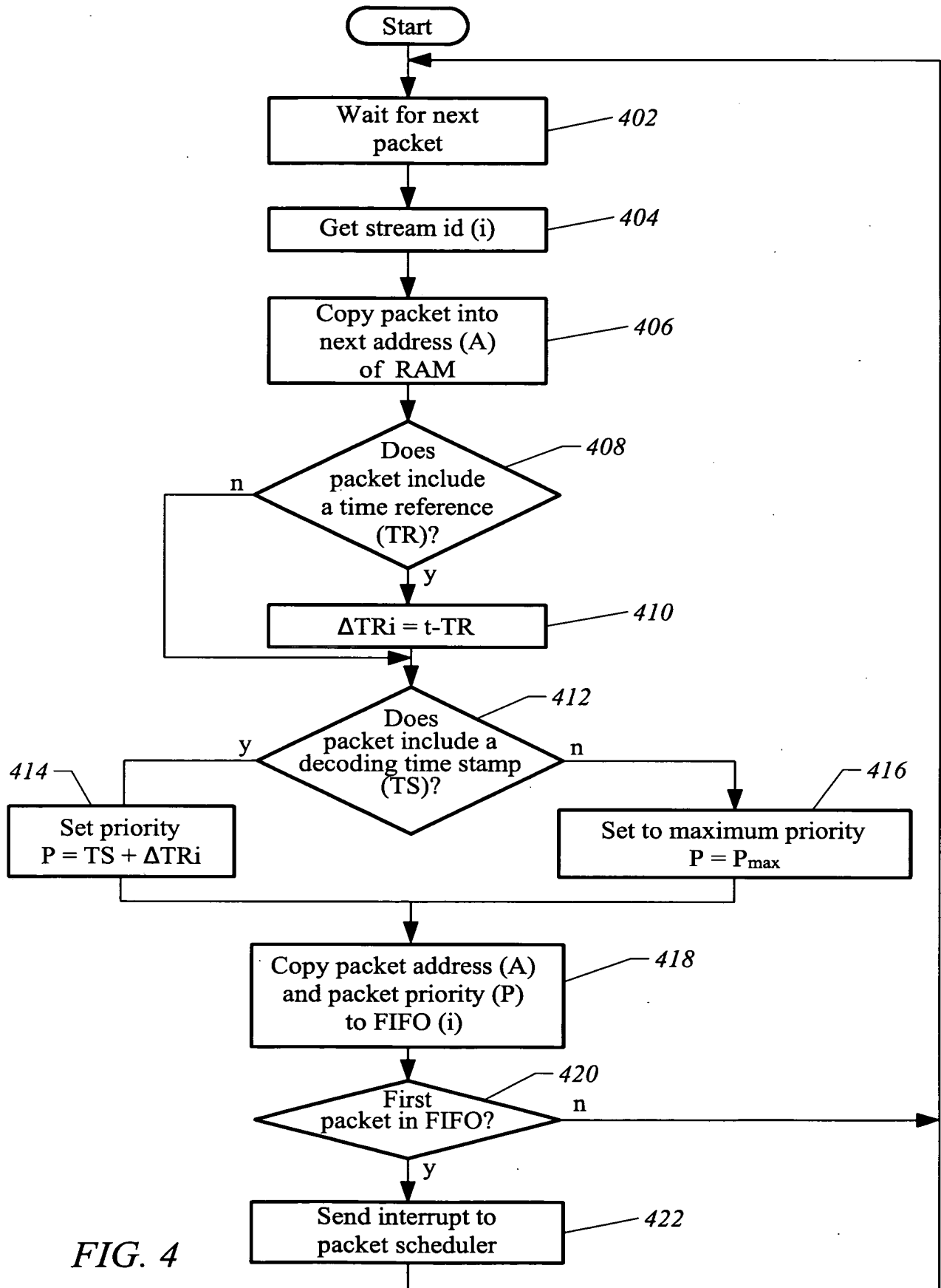
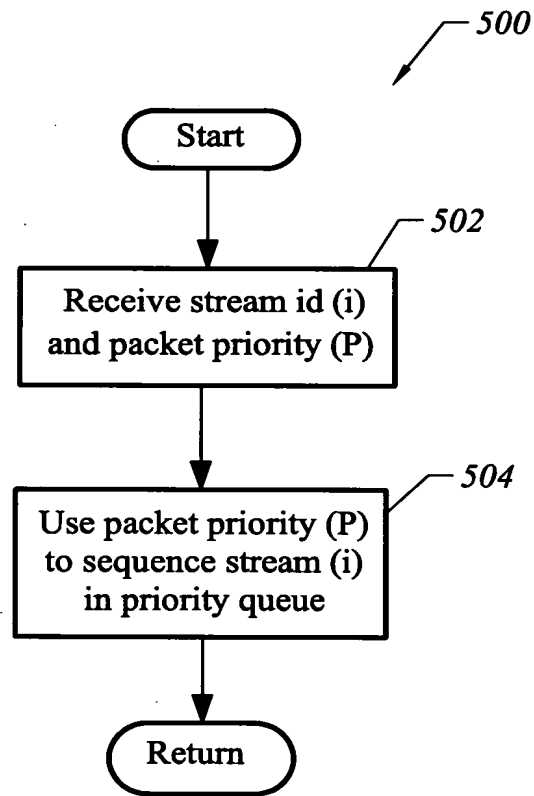


FIG. 4

*FIG. 5*

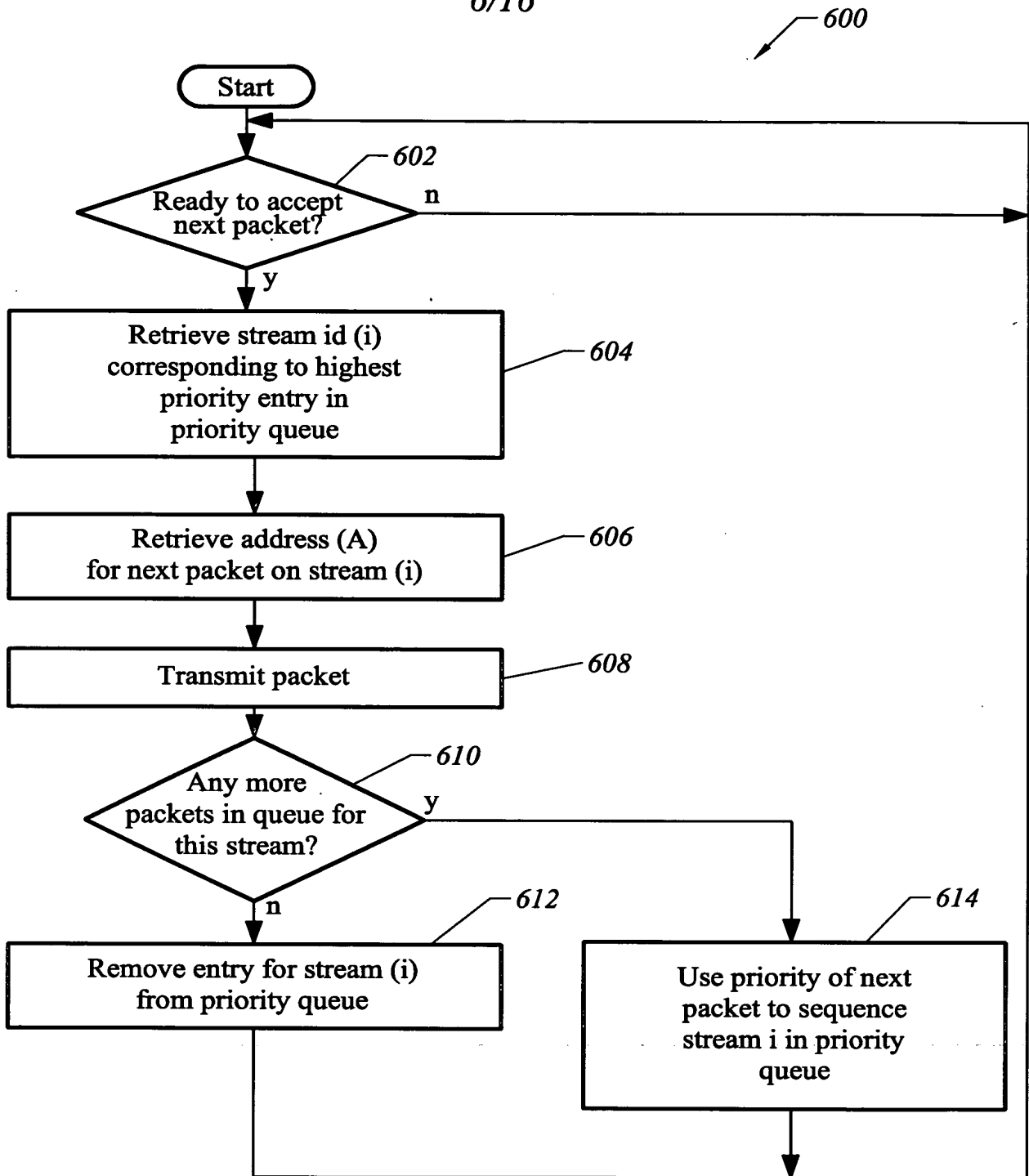


FIG. 6

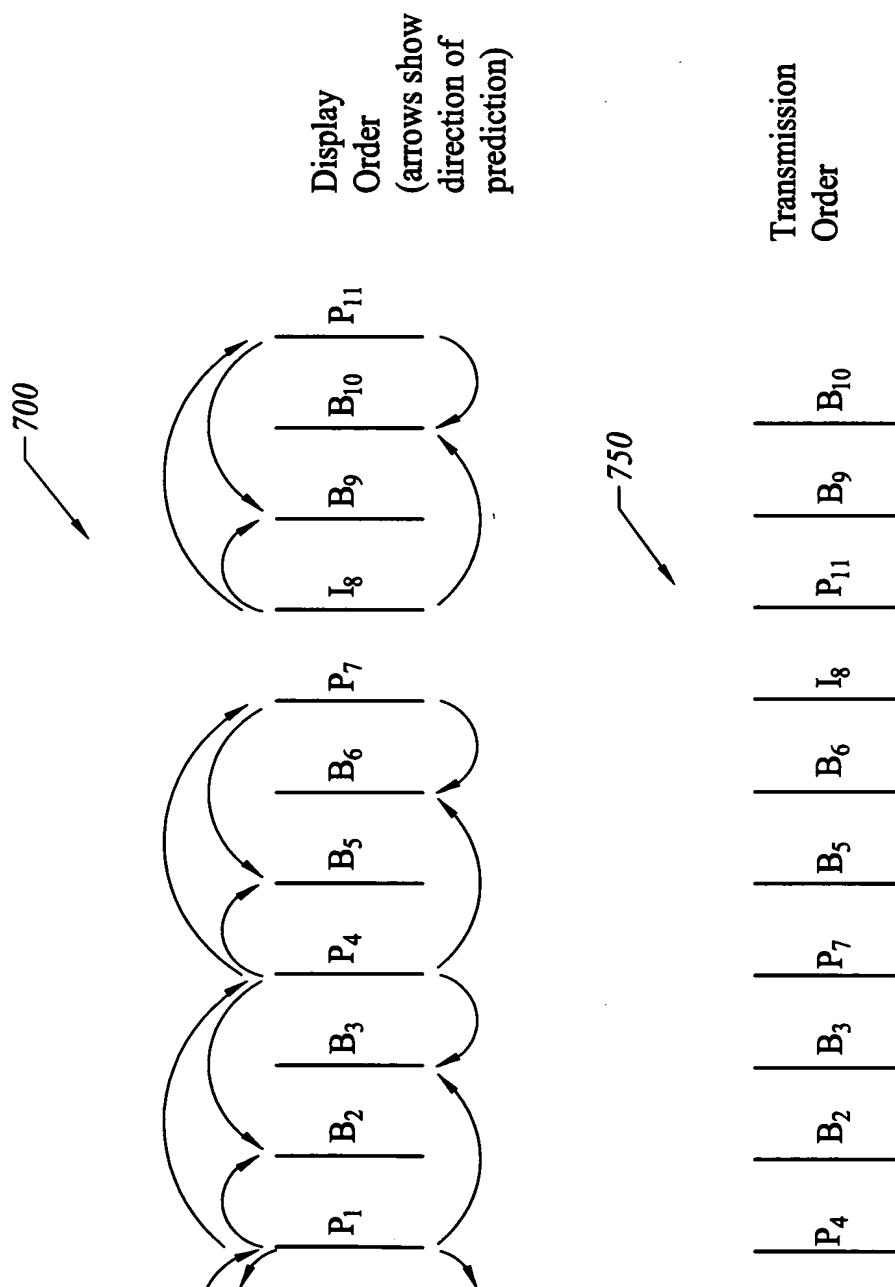
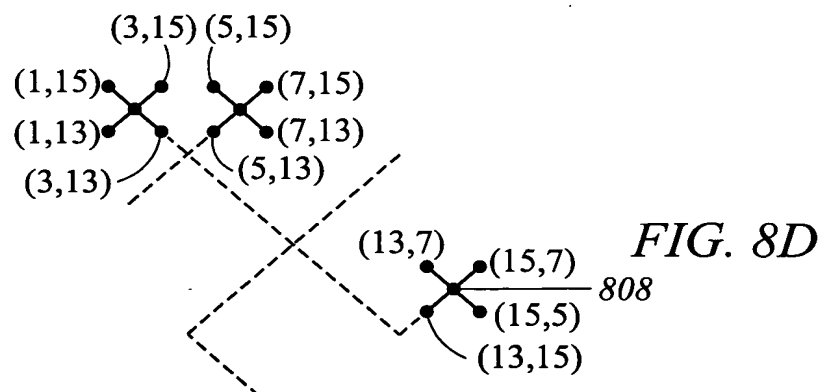
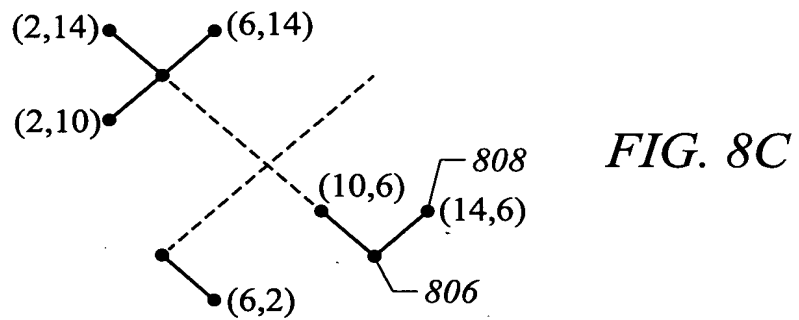
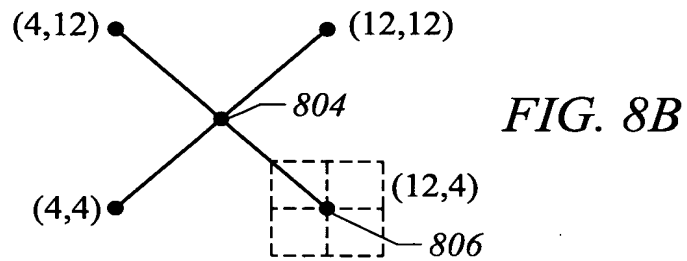
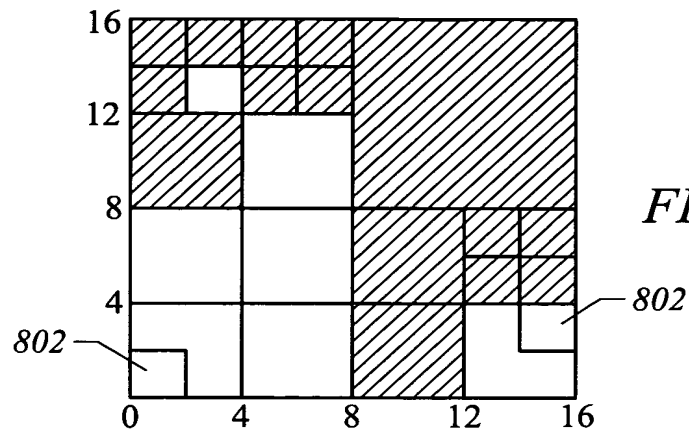


FIG. 7

8/16



```

mem_allocate (d, i, j, k) begin
  if (d > k) begin
    D(i, j) = 0
    return (addr(i, j))
  end
  k=k / 2
  if (d <= D(i+k, j+k) and
      (d > D(i+k, j-k) or D(i+k, j-k) >= D(i+k, j+k)) and
      (d > D(i-k, j+k) or D(i-k, j+k) >= D(i+k, j+k)) and
      (d > D(i-k, j-k) or D(i-k, j-k) >= D(i+k, j+k)))
    a = mem_allocate( d, i+k, j+k, k)
  else if (d <= D(i+k, j-k) and
      (d > D(i-k, j+k) or D(i-k, j+k) >= D(i+k, j-k)) and
      (d > D(i-k, j-k) or D(i-k, j-k) >= D(i+k, j-k)))
    a = mem_allocate ( d, i+k, j-k, k)
  else if ( d <=D(i-k, j+k) and
      (d > D(i-k, j-k) or D(i-k, j-k) >= D(i-k, j+k)))
    a = mem_allocate( d, i-k, j+k, k)
  else
    a = mem_allocate( d, i-k, j-k, k)
  D(i, j) = max( D(i+k, j+k), D(i+k, j-k), D(i-k, j+k), D(i-k, j-k))
  return ( a )
end

```

FIG. 9

```

mem_free ( i, j, k ) begin
  D(i, j) = 2 * k
  while ( k < MEMSIZE/2 ) begin
    k = k * 2
    D(i, j) = max( D(i+k, j+k), D(i+k, j-k), D(i-k, j+k), D(i-k, j-k))
  end
end

```

FIG. 10

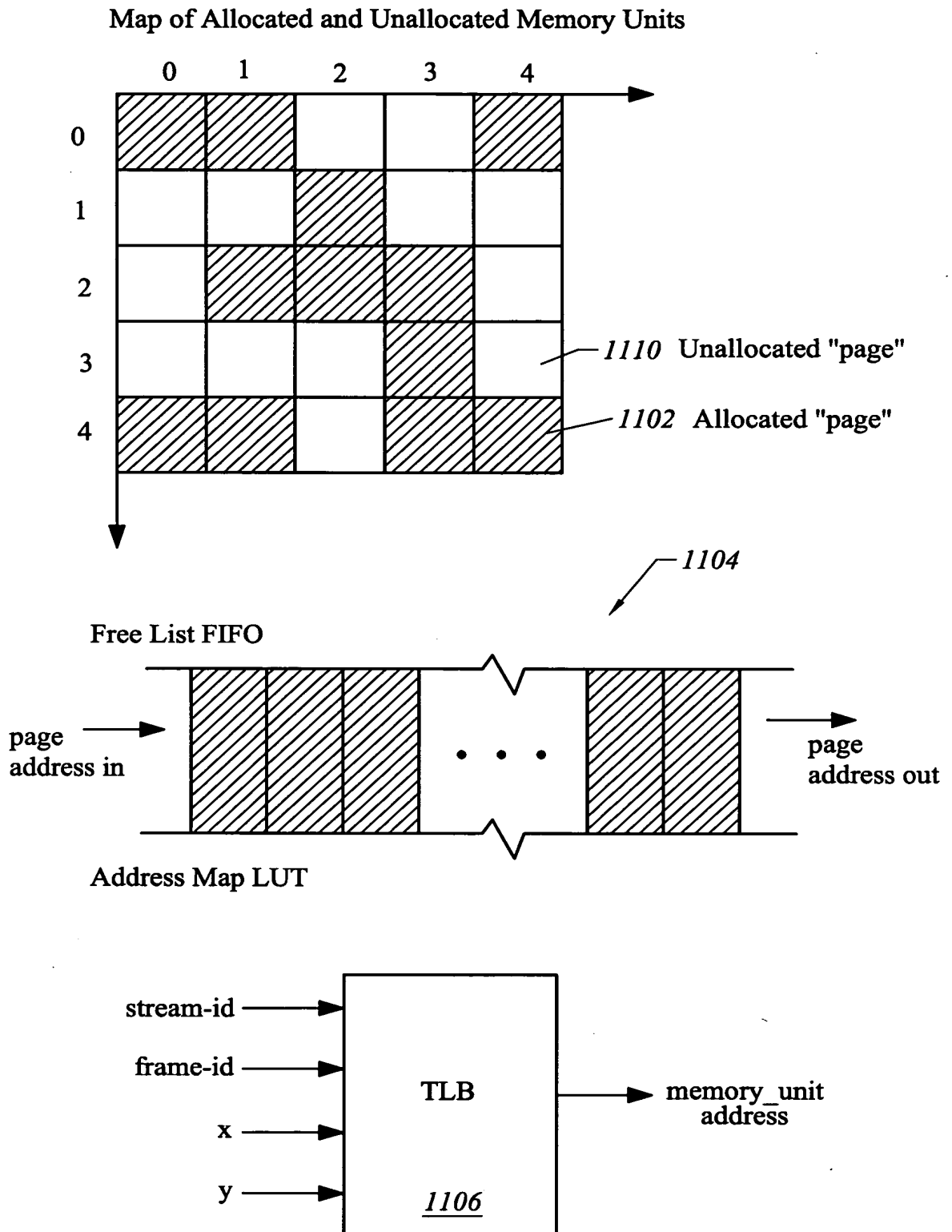


FIG. 11

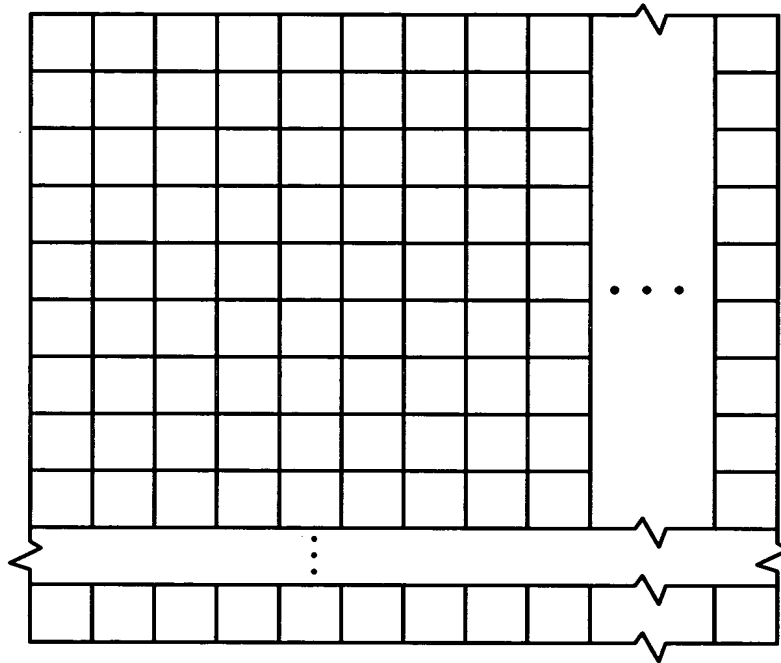


FIG. 12

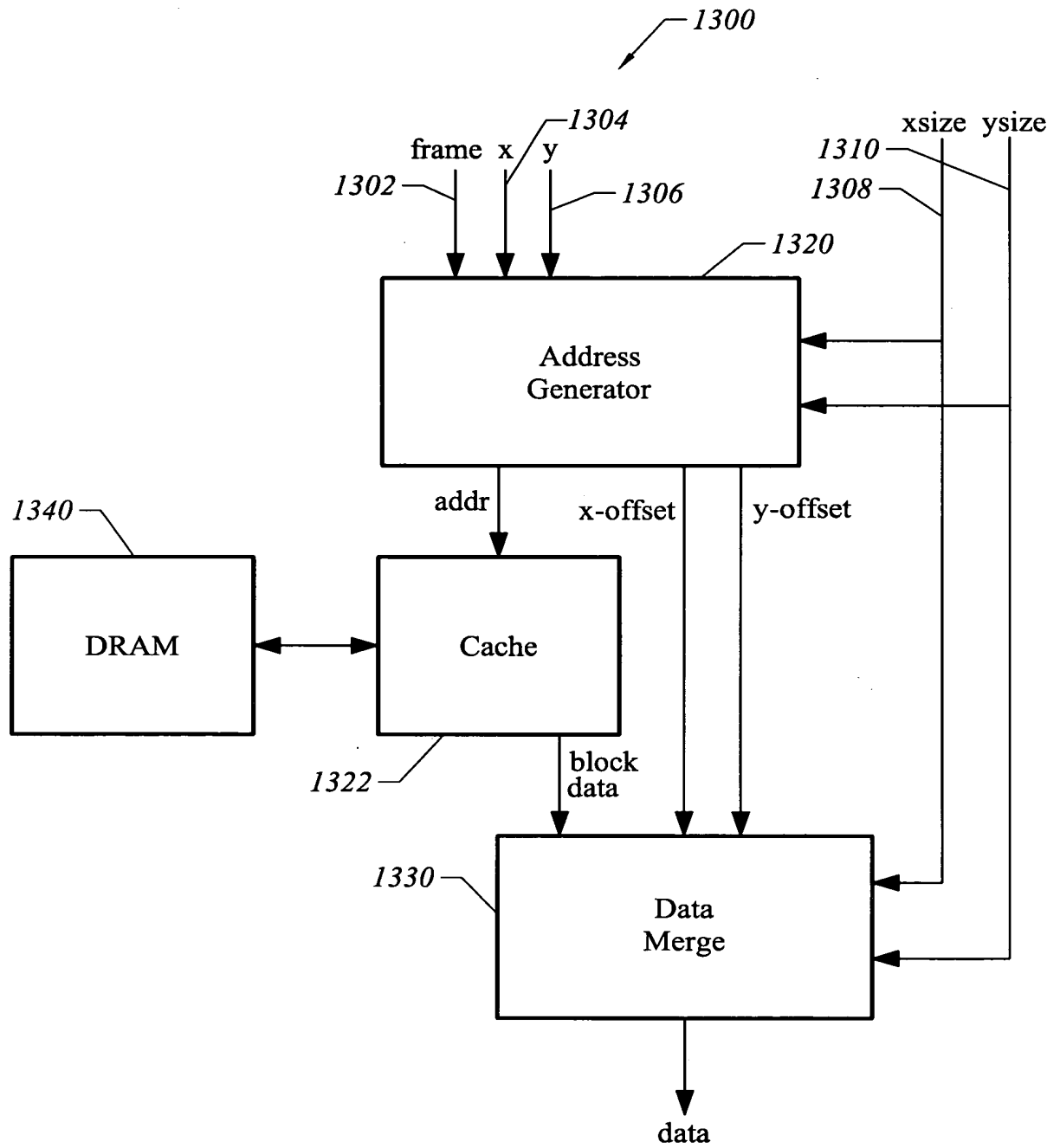


FIG. 13

```

address_generator ( ) begin
    m = 0
    n = 0
    input ( frame, x, y, xsize, ysize)
    while (n < ysize) begin
        x = xaddr + m
        y = yaddr + n
        block_addr = LUT { frame, y[ :7 ], x[ :7 ] }
        y_suboffset = y[6:4]
        x_suboffset = x[6:4]
        addr = { block_addr, y_suboffset, x_suboffset }
        y_offset = y[3:0]
        x_offset = x[3:0]
        output (addr, y_offset, x_offset)
        m = m + 16
        if ( m >= xsize ) begin
            n = n + 16
            m = 0
        end
    end
    return
end

data_merge ( ) begin
    input ( x_size, y_size, x_offset, y_offset)
    n = 0
    while ( n < y_size ) begin
        i = 0

```

FIG. 14

```

while ( i < 16 ) begin
  m = 0
  while ( m < (x_offset + x_size) ) begin
    j = 0
    while (j < 16) begin
      input ( block_data )
      B[i][m] = block_data
      m = m + 1
      j = j + 1
    end
    end
    i = i + 1
  end
  if ( y_offset > 0 ) begin
    i = y_offset
    y_offset = 0
  else
    i = 0
  end

  while ( i < 16 and n < ysize) begin
    while ( j < x_size ) begin
      data = B[i][j + x_offset]
      output ( data )
      j = j + 1
    end
    i = i + 1
    n = n + 1
  end
end
end
end

```

FIG. 15

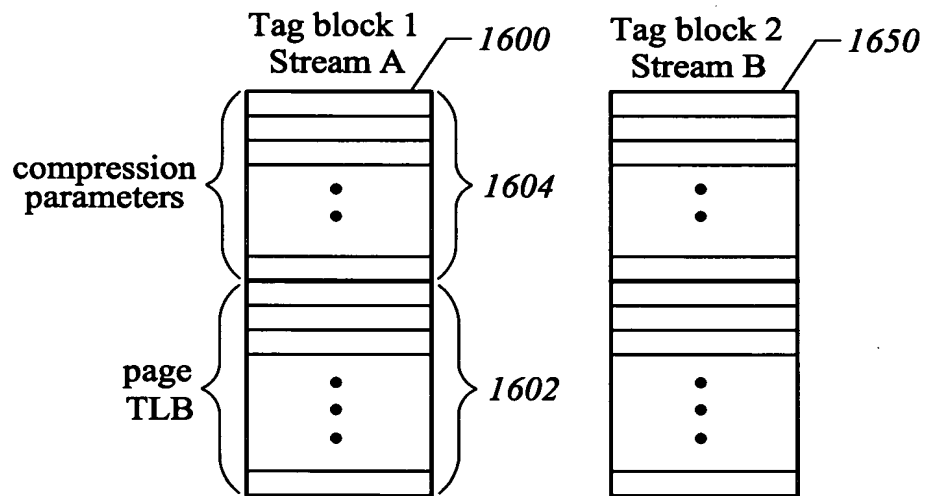


FIG. 16

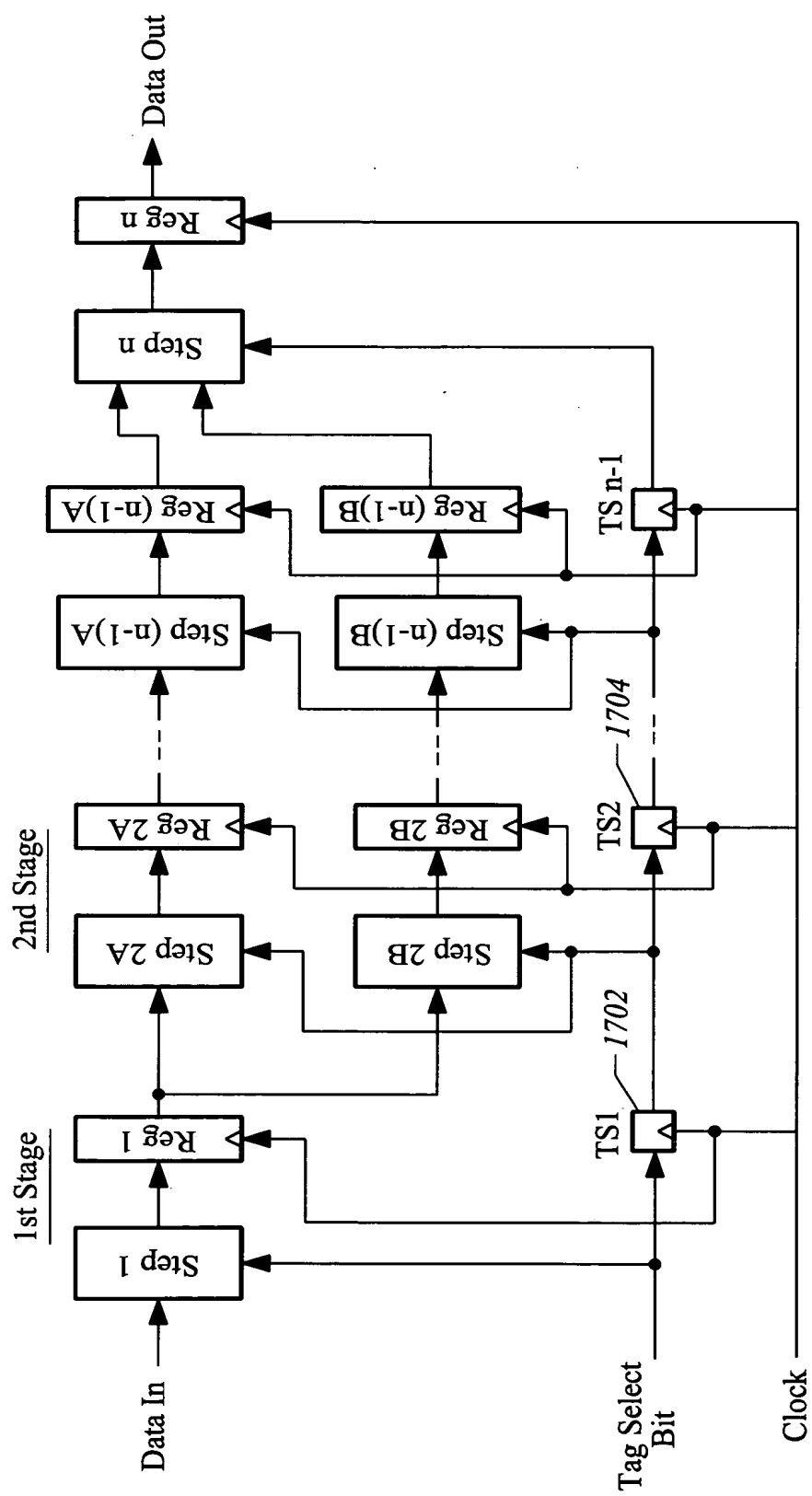


FIG. 17